Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2011
August 29-31, 2011, Washington, DC, USA

DETC2011/ DETC2011-48105

CAPTURING DESIGN PROCESS INFORMATION IN COMPLEX PRODUCT DEVELOPMENT

Krijn Woestenenk*
k.woestenenk@ctw.utwente.nl
G. Maarten Bonnema
Design Engineering
Faculty of Engineering Technology
University of Twente
Enschede, 7500AE
The Netherlands

Andrés A. Alvarez Cabrera
a.a.alvarezcabrera@tudelft.nl
Tetsuo Tomiyama
Intelligent Mechanical Systems
Faculty of 3mE
Delft University of Technology
Delft, 2628CD
The Netherlands

ABSTRACT

From interviewing developers and analyzing examples from industry, the authors have concluded that communication issues during the design process are a key factor of the complexity of product development. These communication issues stem from a lack of insight in the workflow between designers and their resources, and the lack of insight in the relation of this workflow to the system architecture. To the best knowledge of the authors, currently there are no suitable models and tools that allow capturing and understanding such information in an integrated way. This work contributes by providing requirements for tools and models, and proposes a modeling language that fulfils such requirements. With this language we introduce a method for capturing design process information: The language can combine multiple stakeholder-based views on their system aspects of interest with architectural concerns, and can specify which resources in terms of models and parametric information are needed from other stakeholders to develop these aspects. The language was also developed as a stepping stone for automation of design processes.

INTRODUCTION

As we can see around us nowadays, increasingly complex products that provide more functionality (hybrid cars, smart phones, computers, etc.) are being designed and produced routinely in industry. As functionality increases, more and more engineering disciplines and people are needed to develop a product. This phenomenon gives rise to an alarming trend in the development companies: the large number of people and disciplines involved in design leads to an increase in the complexity of the design process that cannot be easily understood with the current tools and methods – failures propagate undetected until production and resources have to be used inefficiently to fix the apparently "last-minute" problems.

Better understanding of the developed system or product (e.g., through the recognition of a product architecture) can increase the awareness of developers regarding design changes and iterations, but may not be sufficient to create an overview that can support the product design activities.

Thus, the design process itself should also be represented in the architecture. To provide a proper underlying framework that supports improvement of design processes by increasing their understanding, we have to state what kind of information has to be modeled, how stakeholders can use, make and change the models, and where and how it is possible to automate tasks.

Therefore, as shown in Figure 1, we propose a stakeholder-centered method to capture architecture-level information (using an adequate language) and a framework to organize such information.

* Corresponding author

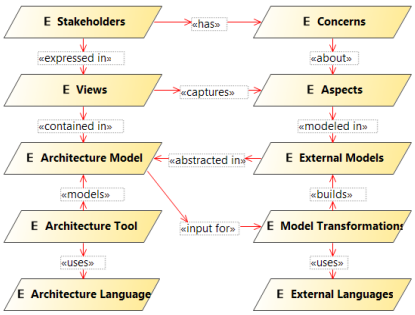

FIGURE 1. MODEL DESIGN PROCESS INFORMATION AROUND THE STAKEHOLDERS.

The proposals in this work are motivated by a series of interviews with stakeholders (in this paper the word stakeholder will be used for anyone interested in or involved in the design process, such as designers, engineers or managers) from several companies during workshops and direct interaction through projects. An overview of the topics and participants of these interviews is shown in table 1. During these interactions, system designers and architects made statements such as:

– "We could not start reasoning about it (the architecture) until we made a picture together."

– "Our whole system development depends on only ten key requirements, they are everywhere; we call them 'Diamonds'."

– "When discussing the system's performance at a meeting, I thought, 'didn't we encounter and fix this problem in the previous version?'."

The main cases involve two companies that attempted to build models to understand their design processes and improve them. More specifically, besides understanding the design process, the goal was to try to decrease the manual hand-over of information (automating where possible by developing new tools) in order to reduce transcription and consistency errors, as well as the effort of these "information transfer" activities. Company A successfully captured the information and improved its design processes, while company B failed to build the model and to globally improve its design processes.

In the first section of this paper, we describe related industrial design practices, discuss the sample cases of companies A and B (and other interviews), and extract the reasons for success or failure to capture the design process, pointing out requirements (desirable characteristics) for models that support capturing design process information. The next section contains a more theoretical analysis of the requirements (this is the first contribution of this work), followed by proposing a model, a framework, and a method fulfilling the requirements which allow capturing the design process information (the second main contribution). The paper finishes with a section about conclusions and future work. Words in **bold** through the document indicate the concepts from a design language used in this paper.

TABLE 1. OVERVIEW OF INTERVIEWS.

Workshops and research project feedback meetings	
participants	~60 people; professors, PhD students, industrial system architects, engineers, technology directors
concerns	Consensus making, Building and communicating system architectures, Communication and versioning of design decisions and design knowledge.
Conferences and academic discussions	
participants	~30 people; (assistant) professors, PhD's, PhD students
concerns	System engineering and architecting, Model & knowledge based engineering, Mechatronics design, Human factor in communicating complex design information, meta & reference modeling.
Company Interviews and case studies	
participants	~40 people; Engineers and managers in multinational high-tech companies: ASML, Mathworks, Océ, Philips, Vanderlande
concerns	Multi-view system architecture, Design decisions, versioning and communication, Model & knowledge based engineering, Consistency in design information, Innovation vs. reuse.

CURRENT INDUSTRIAL PRACTICE

The following paragraphs cover general aspects and interactions of the design processes in current industrial practices. In principle, these processes can be followed and mapped to make models that allow understanding and analyzing the design processes in a specific situation. The next subsection presents two cases of such attempts. From these cases and other experiences we derive a set of requirements for the languages that should allow representing architecture-level information for the purpose of modeling a design process.

Many companies use a central representation of their system to support discussions about the design process across disciplines. This central representation is a company specific view, mostly based on either the system's structural decomposition, with (abstract) **entities** such as components or class instances, or the system's **functional** decomposition, or a mixture of the two. This representation is often based on previous system versions, and as such forms an architecture-level description of the most important and least changing parts of those systems.

A set of **requirements** is put in place to define the design space for the new system. These requirements are translated into local constraints for the various design disciplines, after which a set of models is made (**design tasks**) for the various **aspects** of the system and the system behavior. These models contain **domain entities** that reflect system properties in a certain (engineering) domain, such as the 3D geometry/manufacturing models, (embedded) software, simulation, cost calculation, etc. An aspect groups any information related to the system which a stakeholder or designer is interested in. In that way, aspects can contain

information of multiple domains. An aspect can be captured with a model that represents the stakeholder's **view** on the system. The models contain a number of (tacit) design **parameters** whose values can be calculated, chosen and optimized to match the local constraints of the model and, as such, embody a point in the design space.

The models that represent an aspect are evaluated. Based on the results, design iterations between **design tasks** follow towards the achievement of the global requirements through a mechanism of hand-over and negotiations between the various stakeholders. The hand-over of information can be model-based (e.g., CAD file, set of solved equations, drawing), textual (e.g. Microsoft Visio, Word, Excel [1]), or verbal. Some of the "final" design models are used as input for manufacturing, released as compiled software, or analyzed and stored for reuse purposes. Especially for software engineering, change management or project management tools (such as SAP[2] or IBM Rational Jazz [3]) enable the design teams to work on big projects through shared and reusable models. These shared models are kept consistent by storing changes in the code that makes up the model, and building test suites on top of these models. In other engineering disciplines (e.g., mechanical engineering) models are often reused based on the domain specific design artifacts (i.e., domain entities) rather than the code of the model, which leads to toolboxes of template solutions, building blocks or knowledge bases for the next system versions.

Two industrial cases of design process modeling

Companies A and B decided at some point to get information about their development processes in order to solve some rising problems of miscommunication and make some improvements. At first, capturing the design process appears to be straightforward, but in practice that task proves to be more difficult. The task mainly consisted of interviewing fellow stakeholders and constructing the "information flow" networks. Finding no specialized tools for the task, the people attempting to build such models used common tools (Powerpoint, Visio, etc. [1]) to construct graphical descriptions to represent the information flow as networks of input-process-output boxes.

Company A took a single engineer (on his own initiative) who interviewed his colleagues and ended up with two A0-size posters, covered with interconnected post stamp-size blocks. The posters were then used as a wakeup call to management to do something about the complexity of the design process. Over a period of ten years they incrementally institutionalized a department for design tooling to automate and simplify the design process. Because of this, they can now make designs within a few hours by reusing knowledge from component "libraries". Of course, that does not include the time to develop new functionality.

Company B started with the initiative of a single engineer who studied a "local" design process for his department and (similarly to the engineer of A) ended up with a single A0-size poster, covered with interconnected post stamp-size blocks. The reaction of management after seeing the resulting poster was similar to the one of company A. However, the subsequent approach for action was different: This company made a team to address a tooling and automation project, and gave them a deadline of a few months. As a result, some processes were improved, but every member of the team chose their own solution locally, implementing languages, methods, and deliverables. After the project, the process was halted and people returned to their normal jobs. The resulting improvements (consistency and speed) in automation led to more design freedom, which was used to make more versions of the tooling by individual engineers. The communication became exponentially more complex, and the end result is that they are no better off now than when they started.

Reasons for the long-term success of company A and the only short-term success of the company B may be the lack of an underlying framework to improve the design process and carry tool development. The next section covers these issues in detail.

Issues related to the current practices

This section states a number of issues that lay at the basis of growing complexity in multi-domain design processes. Why is communication in a design process such a problem? What are the reasons of these problems? These issues were identified during workshops and project feedback meetings, conferences and academic discussions, company interviews and case studies with various people associated with complex designs (*cf*. Table 1). We refer the reader again to the statements in the introduction. Further material supporting our conclusions concerning complex mechatronic design processes in industry can be found at [4]. For this paper, these issues can be categorized roughly as architectural problems of creating a context for stakeholders, communication problems of multi-domain cooperation and decision making, and domain problems of consistent models and reuse of design information.

Architectural issues

Architecture descriptions are mostly a structural or behavioral abstraction of the system under design. However, the architecture descriptions are often used to communicate the goals and resources of a design process between stakeholders. This can cause a mismatch, since often the resources (in the form of models) are excluded from the architecture description. For example, an engineer needs to develop a control software model for the function 'move from a to b' with the requirements 'overshoot <5%; t < 5s' for the component 'linear actuator 1'. He or she might be interested in where the overshoot requirement comes from, who decided on that value, and why, and what model contains the value of the requirement.

Furthermore, the description often has a single decomposition which is used to identify the interactions (i.e., interfaces) for all stakeholders in the design process. Problems arise when the chosen decomposition does not fit the interests (i.e., the aspects of interest) of those stakeholders. For example, a stakeholder who is interested in the throughput (aspect) of a production system needs to enumerate the throughput information of all components in the system. These components are owned along the architectural decomposition by a number of different stakeholders, who may have no interest in capturing throughput information. As a result, the stakeholder has to query all the other stakeholders about his throughput information.

This is an error-prone process because

– stakeholders can change the information in the meantime,
– information can be manually recorded with errors,
– stakeholders provide wrong or 'over-budgeted' values because of lack of overview.

Over-budgeted values are values that are not optimal for the system, but are very convenient for the stakeholder to keep his or her own solution space as big as possible. Therefore the decomposition of the system must be done along the aspects the stakeholders are interested in. As a result, there needs to be a variety of decompositions. This is further complicated by the observation that the aspects in modern systems are often of a multi-disciplinary nature (real-time behavior, multi-physics finite element analysis, embedded software control, etc.), while the design domains of the companies are often organized according to the 'traditional' domains of mechanical, electrical and software specializations of the employees.

Communication issues

There is no single language, even within a single company, to communicate properly design problems that span over multiple domains. All stakeholders have been educated in their own languages, abstractions and modeling tools. Also, a general sense of abstract thinking seems to be lacking in many of the stakeholders, which further hampers the recognition of shared goals and communication. Many companies therefore develop their own internal design process language, which often exists separated from the system architecture:

– Organizations are decomposed into domains such as 'mechanics', 'electronics' and 'software'.
– Design activities are decomposed in product specific aspects as 'paper path', 'imaging' or 'cost'.
– Systems are decomposed in components and functions.
– Design space is decomposed in budgets of requirements, often based on less than ten key performance indicators, which represent added value to the costumer.

A lack of such cross-domain abstractions can cause a lot of extra work for the stakeholders. This can be made clear by an example. Aspect models can be made with automated design tools that provide ways to verify and validate the model and keep it consistent from version to version. However, manual hand-over of important results of these models between domains or stakeholders creates duplicate information about important design decisions which is difficult to be consistently updated. Hand-over of information between disciplines is done manually or verbally because of the lack of automated tooling that can capture the company-specific language, while at the same time automated tools lack the 'freedom' to capture design architecture.

As a result, design decisions are forgotten or inconsistently applied. Because it is not clear where a design decision comes from, negotiations on interfaces or trade-offs between stakeholders are not supported by data from the current state of the design process. Also, the 'wrong' decisions are not remembered, because the context was not clear, resulting in recurring errors in the design from version to version.

From another perspective, information flow in product development involves many transformations to couple requirements with domains, or domains to each other: for example, cost (€), to man hours (h), to component and tooling cost (€), to bill of material model. These transformations are not formally recorded because there is no clear ownership on them, since they lay in the interface of domains and their stakeholders. This again points to the lack of a common language as the origin of many communication problems.

Domain issues

The models that describe the structure and behavior of the system are made in design tools (e.g. Matlab [5], CATIA [6], Excel [1]) that have been based on accumulated design knowledge and best practice over many years. These models are widely used and very useful, and therefore, it is not our goal to capture all knowledge of these models and make it executable. However, in the context of the design process, we can envision room for improvement.

From our interviews, it became clear that the domain experts do not need the full explicit knowledge of the input for their model. They know what they need to make a model, how to make the model, and what the model needs to deliver. For example, model the geometry of a conveyor, or, model the control algorithm between a sensor and actuator. What domain experts often do not know is the relation of their model to other models in the larger context of the design process. Often, the experts get a text document with a general description of a task.

After this, they search in previous, 'upstream', models to get the parameters (or input data, or constraints) needed to construct their own models. Often the upstream models do not provide the needed data in a consistent format, since the upstream model builder does not need that information for his or her own design task: An engineer of one company measured the geometry of a PDF document print out of a CAD drawing with a ruler to get his input; he did this multiple times in the development of a system. The engineer who made the CAD

drawing could easily expose the needed values in a list, but did not, because it is out of scope of that specific design task.

If the input for their models does not generate a valid solution, the experts may discuss their shared parameters (next to the coffee machine) and update the values to their mutual satisfaction. However, the two experts do not clearly know the context of their decisions, because they do not have access to the global architecture description of the design.

Requirements to capture design process information

The analysis of the problems has provided us with a number of issues that need to be addressed to successfully capture design process information. We enumerate here some requirements for an architecture description that, we think, address the presented issues. Therefore, the architecture description language must provide:

– A limited set of concepts to capture the domain-specific abstractions and decompositions, and their interrelations. This provides clear concepts to use in negotiations and discussions.

– Representation of 'multi-views' and 'multi-mappings', from the functional and structural decompositions, and from the requirements to the various aspects and domain. This helps to individually address to the stakeholders that wish to use an architecture description as support.

– A mechanism to model the design information flow in terms of shared parameters, the interfaces between stakeholder aspects, and their relation to requirements and system functionality. This allows relating design decisions to the context and goals of design.

– A mechanism to define the required input and output parameters of design tasks and models, both inside the models as well as in the larger design context. This allows identifying ownership over design processes and design responsibilities.

– A format for resource information on design parameters, models and users. This allows for exchange of consistent and up-to-date information.

– Letting individual stakeholders globally update the information in the design process, in view of their aspect models, should therefore be supported. This allows maintaining updated information for all stakeholders.

CAPTURING DESIGN PROCESS INFORMATION OF COMPLEX SYSTEMS

This section contains an analysis of the previous section which results in requirements and characteristics of the proposed approach. Then we present a proposal for capturing the design process information by introducing the two main components, i.e., the language to capture architecture information and the method to connect the resources the design process needs (please recall the introduction and Figure 1).

Analysis of the problem and the solution requirements

As is made clear from the discussed issues, the design process is not a clear 'waterfall' down from requirements to a working design. In the process, design decisions are constantly challenged and iterated towards a globally valid solution. This means that the design decisions at one particular point in time have a (large) set of accompanying models. When a decision is changed, the accompanying models need to be changed too, which can cause a cascade in other design decisions. In software development, current versioning and change management tooling works on a 'data level', keeping track of the differences in the data inside the model (v1 = aaaa, v2 = aabbaa), who made the changes, and the ownership of the changes. This could be applied to other design disciplines too.

However, in other disciplines, the changed file date is not the most important change information, as it lacks the reasoning behind the change. Stakeholders need traceability and rationale of design decisions more than the detailed model changes resulting from these decisions.

Project management tooling supports very well time-based management. This is done by using milestones, deadlines, scheduling, and change management in tools like IBM rational Jazz. However, the deliverables of the individual work packages seem to be meagerly defined, and cannot be evaluated in the larger project context. This stems from the same reason explaining manual hand-over (cf. section "communication issues"): there is no single language to exchange such information, or many languages need to be integrated, requiring semantically rich descriptions of the model content [7].

However, if this process is made explicit without capturing the model content, many of these decisions can be captured in architecture versions over time. Furthermore, the decisions can be implemented in new model versions by automatic model transformation and generation. This automation is of course dependent on the availability of explicit design knowledge, which is out of scope of the paper. However, the prerequisites can be made by providing a machine-readable format for reading and writing the architecture description language.

The solution should allow representing the evolution of the architecture information as the design process progresses. Thus, the architecture model must be able to capture abstract architectural information as well as domain specific details, and must be usable throughout the design process. This is depicted in figure 2. This means the design process workflow is separated from the model instances. The architecture models thus capture the models, viewpoints and design decisions at a certain point in time and evolve through the design process as shown in Figure 4.

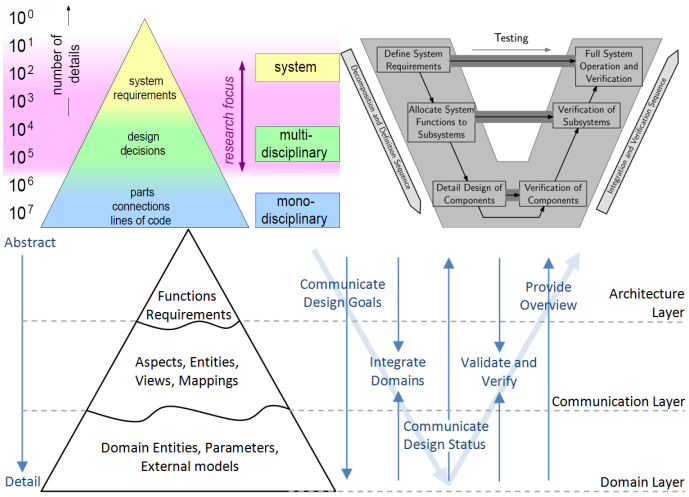

FIGURE 2. CONFRONTATION OF THE ARCHITECTURE – LEVEL INFORMATION (BOTTOM) TO THE PRODUCT INFORMATION [8] (TOP LEFT) AND A VEE DEVELOPMENT PROCESS [9] (TOP RIGHT).

The language should not capture all design knowledge. As this requires too much effort, creates duplicate information, and generates an information flow bottleneck. As depicted in Figure 3, the design information space can be separated in design workflow, knowledge, and architecture. Design workflow is required for project planning, design decisions, and versioning. Design knowledge specifies how to design the system. Design architecture contains the interrelations in a design process in the form of architecture descriptions, models and stakeholder views. Excluding design workflow and knowledge leaves the design architecture. The design architecture model in this context is a reference model for overview and versioning and a meta-model for model generation with a focus on the design process instead of the system. In this model, we capture the minimally needed information an organization needs to make a complex design.

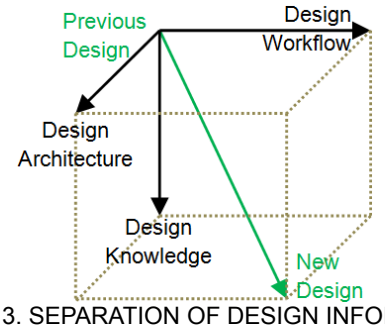

FIGURE 3. SEPARATION OF DESIGN INFORMATION IN ARCHITECTURE, KNOWLEDGE AND WORKFLOW .

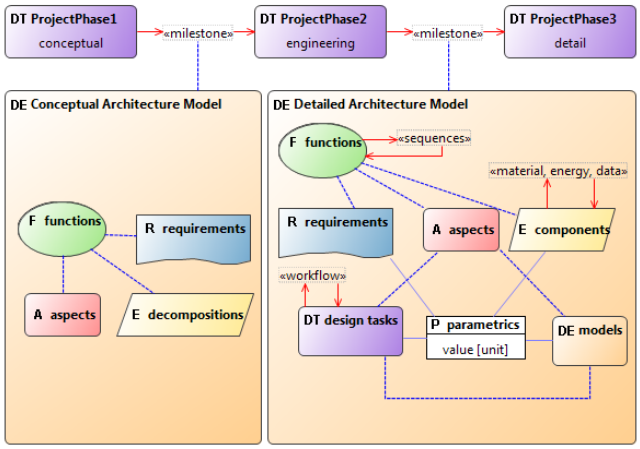

FIGURE 4. ARCHITECTURE EVOLUTION TO FOLLOW THE GROWING AMOUNT OF DETAILS IN THE DESIGN.

Modeling design process information

This section follows the most important issues in the requirements to capture design process information section:

1. the architecture language – A limited set of concepts to capture the domain-specific abstractions and decompositions, and their interrelations. This was discussed in the last section.
2. Setting up architecture models – define multiple views and mappings, from the function and system decomposition and requirements to the various aspects and domain models.
3. Setting up design flow models – Model the design flow in terms of shared parameters, the interfaces between stakeholder aspects, and their relation to requirements and functionality.
4. Setting up knowledge models – A mechanism to define the required input/output parameters of design tasks and models inside the models as well as in the context of the larger design.
5. Keeping track of design resources – A format for resource information on design parameters, models and users.
6. The architecture modeling tool – A tool that lets individual engineers evaluate and update the global design process, in view of their aspect models,
7. Automation – A machine-readable format for reading and writing the architecture models.

Proposed base language to capture architecture-level information

In this section we introduce the language that we need to capture the design process information. The language consists of a set of generic attributes, concept definitions and relation definitions. These concepts have been introduced in another paper [10], and were briefly explained the industrial practices section. Figure 5 and 6 show the classes and relations in the **architecture modeling** language. All concepts are derived from a basic **basicObject** class, which handles the most basic

information content needed in every information object. Furthermore, the **view** class specifies where the concepts are represented. The **parameter** is an atomic concept – used to, for example, store design decisions or parametric design variables – and can be mapped to all other concepts. The other classes are specializations that allow us to make a formal, and thus machine readable, architecture description.

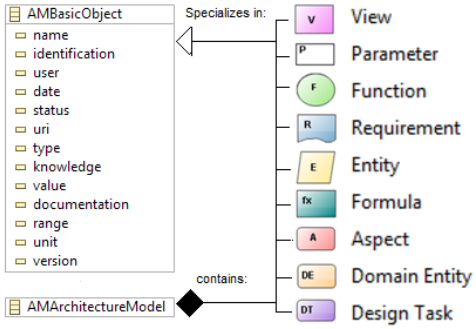

FIGURE 5: BASIC CLASS TYPES IN THE ARCHITECTURE MODELING LANGUAGE.

In a system, **entities** perform **functions** within **requirements**. Other non-functional **aspects** of the system also need to be within requirements. **design tasks** will deliver models containing domain specific **domain entities** that describe the system behavior (functions), structure (entities) and other aspects in the language of that domain. **formulas** can be used to model the math required to calculate parameter values. The black relations are used to model hierarchies or **compositions** within one concept type (functions, requirements, entities, aspects and domain entities) . The blue relations are **mappings** between concept types, used to construct relations between different compositions. The red relations are **flows** between concepts: (temporal) sequence relations between **functions**, information/material/energy flows between **entities**, input/output between **design tasks** and calculation dependency between **parameters**. In order to facilitate modeling the interface between function, entity and design task flows, specific **functionRelation**, **entityRelation** and **designTaskRelation** concepts are used. These extra relation concepts are needed to further specify the sort of interactions the concepts have.

The function relations can be constrained by requirements to set logical conditions for, for example, state transitions, the entity relations can have formulae to specify changes in the modeled flow, and design task relations can have domain entities to specify hand-over documentation, models or specifications.

	BO	AM	V	P	F	R	E	Fx	A	DE	DT	FR	ER	DR	SF
BO		inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	inh	
BO		con	con	map											
AM			inh												
V			com												
P				flo											
F					com	map	map		map			flo			map
R						com			map			map			
E							com		map				flo		
Fx		cross references											map		
A									com	map	map				
DE										com				map	
DT														flo	

Concepts - BO:BasicObject;AM:ArchitectureModel;V:View;P:Parameter;F:Function;
R:Requirement;E:Entity;Fx:Formula;A:Aspect;DE:DomainEntity;DT:DesignTask;
Relation Interfaces - FR:FunctionRelation;ER:EntityRelation;DR:DesignTaskRelation;
Relations - SF:StartFunction;com:(de-)composition;con:containment(parent model);
inh:inheritance;flo:flow(I/O,dependency,sequence);map:mapping(reference)

FIGURE 6: ALLOWED RELATION TYPES IN THE ARCHITECTURE MODELING LANGUAGE.

Setting up architecture models

The language allows for the definition of architecture models based on the views of the stakeholders. This breaks with the tradition of having a system-oriented view, in which the stakeholder must navigate to find information. While the more traditional decompositions of functions, requirements and entities are still important views on the system, a stakeholder can now define a cross-cutting view of the system based on the aspect the stakeholder is interested in. The stakeholder can then define which models are used in developing the aspect, keeping the whole architecture consistent. In figure 7 we see a oversimplified model of a tennis game, where a stakeholder is interested in seeing how a ball will fly after it is hit (bouncing ball view). In order to simulate a single hit of a ball (cf. figure 9), we need to define and share a lot of information, and interpret that in a certain order (cf. figure 8). Shared information across views is depicted by a shortcut symbol left of the concepts, indicating it is defined elsewhere.

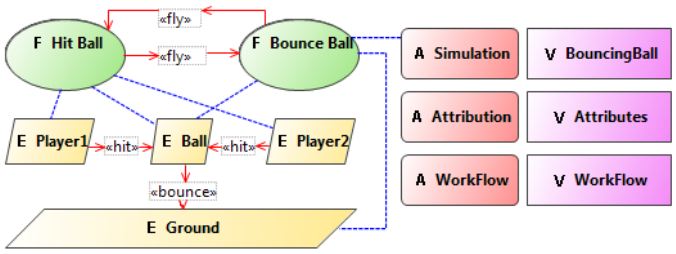

FIGURE 7: STAKEHOLDER CENTERED VIEWS BASED ON THE SYSTEM ARCHITECTURE. THE VIEWS 'WORKFLOW' AND 'BOUNCING BALL' ARE DETAILED IN FIGURES 8 AND 9.

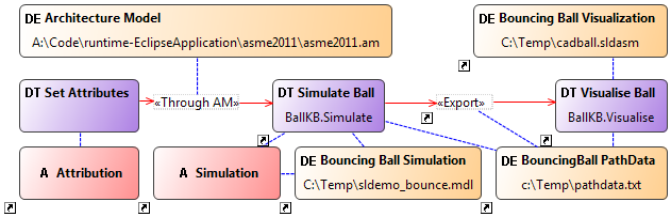

FIGURE 8: DESIGN PROCESS WORKFLOW VIEW, INCLUDING THE ASPECTS AND DOMAIN MODELS.

Setting up design flow models

The design flow can be modeled using design task concepts. A stakeholder performs a certain design task, by reviewing and producing models (domain entities). If we go back to the tennis example, we can define a design process view, that looks at the system from a workflow point of view (cf. figure 8). In this case you need to first define values of the system attributes in order to perform a simulation. With the resulting simulation data, you can then visualize the ball's path.

Setting up knowledge models

What information does an expert really need to develop his model? If we assume he or she already has the knowledge and tools to make the model, as explained above, we can see that model development as a black box within the design space, which needs to be constrained by attributing values to parameters, and which delivers some results. In the bouncing ball example of figure 9, we see that three shared parameters are enough to constrain a shared CAD and simulation model.

This is possible because most third party modeling software have either a text based input/output on which model transformations can work (e.g. Matlab .mdl files) or they have advanced application programming interfaces, through which the tool's functionality is accessible directly (CATIA API). Packaging such knowledge into dedicated design tasks along the line of multi-representation architecture (MRA) patterns for modeling & simulation can then be taken as a next step [11].

Making these model transformations is a knowledge based activity, and is out of scope of the paper – however the ball example shows that the architecture language facilitates such transformations, of which further industrially relevant experiments can be found in [12].

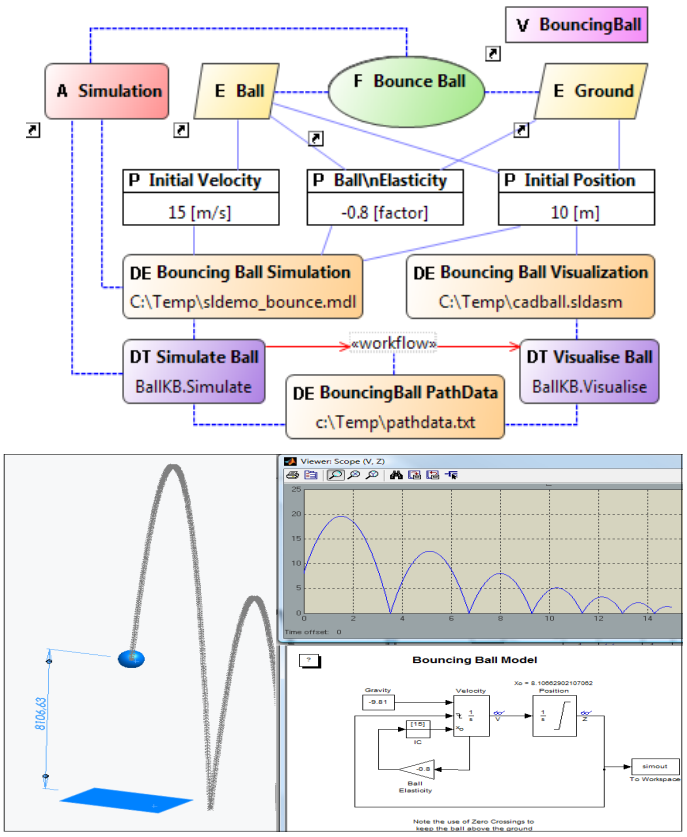

FIGURE 9: MAPPING ABSTRACT ARCHITECTURE INFORMATION TO DETAILED DOMAIN MODELS, AND RUNNING SIMULATIONS THROUGH DESIGN TASKS.

Keeping track of design resources

This information is given partly in the architecture model mappings, and partly in the standard attributes of every concept (cf. figure 10 for an example). External resources can be referenced in the 'uri' attribute, which can contain uniform resource identifiers (URI). User information is kept up-to-date by storing 'user' and modification 'date' information in every concept. This makes sure that the ownership of the information is always known, so other users can negotiate or discuss the information in question. Parametric information is stored in the parameter concepts, where attributes such as 'value', 'type', 'range' and 'unit' can be used to exchange and use information consistently. Furthermore, a 'status' attribute can be kept to follow the progress of the design information. This information can be synchronized to external models using dedicated design tasks as discussed in the 'setting up knowledge models' section.

Property	Value
Date	5/25/2011 10:46:30
Dependencies In	
Dependencies Out	
Documentation	Combines the adsorption of energy when a ball hits the ground
Identification	c93f729e-f34c-4997-bf5a-22541135407a
Knowledge	
Map To AM Views	AM View BouncingBall
Map To Basic Objects	AM Domain Entity Bouncing Ball Simulation, AM Entity Ball, AM
Map To Parameters	
Name	Ball\nElasticity
Range	{-1..0}
Status	OK
Type	Factor
Unit	factor
Uri	
User	Krijn Woestenenk
Value	-0.8
Version	1.0

FIGURE 10: INFORMATION OF AN IMPORTANT SHARED PARAMETER. IT IS MAPPED TO TWO ENTITIES AND TWO DOMAIN ENTITIES, GIVES THE RATIO WITH WHICH ENERGY DISSIPATES WHEN A BALL HITS THE GROUND.

The architecture modeling tool

The architecture language is implemented in the Eclipse Graphical Modeling Framework [13]. This open source project enables the definition of Domain Specific Languages, and implements this language in a graphical editor. The editor allows for the definition of the diagrams that are shown throughout this paper, in a way that is very similar to other tools such as Visio. The benefit is that the language definition constrains the syntax of the models, so that only allowable relations can be made, and consistency can be assured by containing all views and their content in an interrelated model. When one concept is changed, it changes in all views, this allowing evaluation and updating of information throughout the design process and in all connected aspects. Two important aspects were implemented in the Architecture Modeling Framework:

First: The flat model – The information is contained in a flat 'list'. All hierarchical, dependency, and mapping information is then stored in references on the concepts instead of in the data-file hierarchy. This makes it possible to make the views stakeholder-centered: Instead of pre-defining views for functions, requirement tracking, domain models, etcetera, and then let stakeholders find their way around them, we define views for the stakeholders interest, and let them represent the functions, requirements, etc. from the flat model that they need. The hierarchies are still there, and a specific view for functional decomposition or requirement decomposition is still convenient, but these views have no special status per definition. In figure 7, we see three views that are related through their concepts.

Second: The extendibility (cf. figure 11) – The Eclipse framework allows for layering of tooling on top of each other – starting with a language definition (which is a class diagram) – to a graphical editor – to tooling for consistency, querying, databasing, exporting/importing, model reuse etc. – to knowledge based tooling that can actually manipulate the models with explicitly programmed design knowledge.

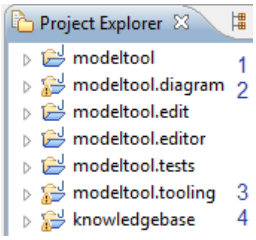

FIGURE 11: EXTENDABILITY OF THE ECLIPSE PLATFORM. 1. DEFINES A LANGUAGE, 2. DEFINES THE EDITOR, 3. DEFINES GENERIC INFORMATION TOOLING, 4. DEFINES KNOWLEDGE BASED TOOLING.

Automation

An important requirement for our architecture modeling framework was the usability of our tool in the direct application of knowledge based engineering. This topic deserves its own paper and is handled in the sister paper of this one in [12].

RESULTS AND DISCUSSION

Our method for capturing information for complex product development is setting up an architecture model from the stakeholder point of view. Therefore we enable modeling the aspects the stakeholder is interested in on three levels:

1. Architecture: How is the system affected by this aspect? Because these types of information are highly interwoven, we define the architecture models along a formal relational language (cf. figures 5 and 6). This language separates the architecture model hierarchies from the hierarchies of the stakeholder-centered information in such a way that new views can be cut across the existing information (cf. figure 7 to 9). This integral approach makes it possible to model various stakeholder views consistently inside one model.

2. Knowledge: What information is needed to construct or evaluate models about this aspect? Because we focus on the communication between stakeholders – rather than a complete structural and behavioral description of the system – we can exclude the ontological information of the system and the models and tools that describe the system (cf. figure 8). This means this language is not very useful in coming to an initial understanding of how and why the system does what it does. However, it does provide a simple way to codify the resulting architecture description. This keeps the language simple enough to express the workings and decompositions of a system in a single model, and expressive enough to act as a starting point for model generation and evaluation (cf. figure 9).

3. Workflow: Who has ownership of this information? Where is the up-to-date version of this information? What results of the models have to be recorded? Because we model the information exchange between stakeholders directly in terms of design tasks and design decisions, many manual

model transformations, in the form of intermediate Excel or Word or Visio documents, should become superfluous.

As seen in the 'Issues related to the current practices' section, these manual documents take up many man-hours, while small errors in these documents tend to propagate into very large errors in terms of system behavior. Consistency of information is one of the most important reasons of applying the architecture model.

CONCLUSION

We have shown that only a few concepts (such as **parameter**, **requirement**, **aspect**) are needed to describe complex design process information. With these concepts we model the objectives of an aspect model or a stakeholder in the design process rather than what a system is or does – we do not model the system. Choosing the right concepts in a design language separates concerns consistently, and as such can reduce perceived complexity in communication across disciplines. Because by mapping these concepts together again in a formal way, the resulting reference model gives the user the context he/she needs to better evaluate the design process. The same formalisms make it possible to facilitate automation of parts of the design process.

The stated issues in complex design are not resolved by looking at a subset of solutions for 'only' the architecture description, 'only' the workflow or 'only' the design knowledge automation, it cuts across all these domains. The authors have tried to develop the architecture language in order to facilitate system architecting and engineering in an integrated way.

The proposed method is not complete. This work identifies the necessary information to build the model of a design process, but it does not provide a method to construct such a model from the presented information. Future research is needed, mainly on the topics of knowledge automation, and capturing design decisions in workflow models. Further refinement of the concepts and relation types should be made through additional industrial case studies.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of the Innovation-Oriented Research Programme 'Integral Product Creation and Realization (IOP IPCR)' of the Netherlands Ministry of Economic Affairs, Agriculture and Innovation.

REFERENCES

[1] Word, Excel, PowerPoint and Visio are registered trademarks of Microsoft Co.

[2] SAP is a registered trademark of SAP AG.

[3] IBM Rational Jazz is a registered trademark of IBM Co.

[4] Jackson, C.K., 2006. *The Mechatronics System Design Benchmark Report – Coordinating Engineering Disciplines*. Boston, MA, USA: Aberdeen Group.

[5] Matlab is a registered trademark of Mathworks, Co.

[6] CATIA is a registered trademark of Dassault Systèmes.

[7] Gerritsen, B.H.M., Gielingh W., Dankwort, C.W., Anderl, R., 2011. "Frameworks and technologies for exchanging and sharing product life cycle knowledge". *Computer-Aided Design*, volume 43, number 5, p459-463.

[8] Bonnema, G.M., 2008. *Funkey architecting : an integrated approach to system architecting using functions, key drivers and system budgets*, Thesis. University of Twente, The Netherlands. See also URL http://doc.utwente.nl/58868/.

[9] Muller, G.J., 2010. *Design Objectives and Design Understandability*, See also URL http://www.gaudisite.nl/DesignUnderstandabilitySlides.pdf.

[10] Alvarez Cabrera, A.A., Woestenenk, K., Tomiyama, T., 2011. "An Architecture Model to Support Cooperative Design for Mechatronic Products: Control Design Case", *Mechatronics*, doi: 10.1016/j.mechatronics.2011.01.009.

[11] Peak, R.S., Burkhart, R.M., Friedenthal, S.A., Wilson, M.W., Bajaj, M., Kim, I., 2007. "Simulation-Based Design Using SysML—Part 2: Celebrating Diversity by Example". *INCOSE Intl. Symposium*, San Diego.

[12] Alvarez Cabrera, A.A., Foeken, M.J., Woestenenk, K., Stoot, G., Tomiyama, T., 2011. "Modeling and using product architectures in industrial mechatronic product development: experiments and observations". Proceedings of the *ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*. Washington, DC, USA

[13] Eclipse Graphical Modeling Project, 2011. See also URL http://www.eclipse.org/modeling/gmp/.